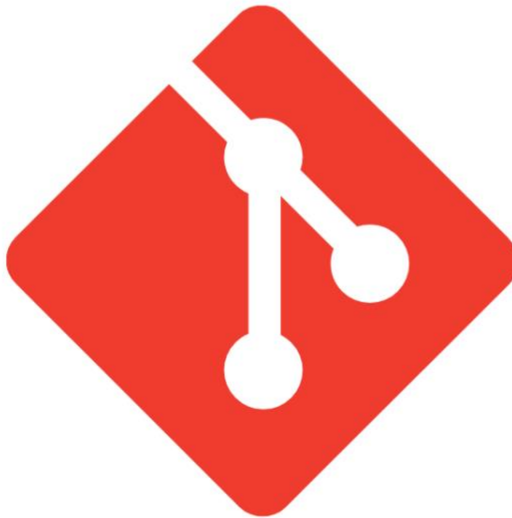
# تعلّم

# git

(الأداة الأكثر استعمالاً لإدارة الإصدارات)



$ > git commit -m "م. مختار سيّد صالح"

# تعلّم git

---

(الأداة الأكثر استعمالاً لإدارة الإصدارات)

تأليف:

م. مختار سيّد صالح

*الفصل الأوَّل*

# أساسيّات git

## ما هي git؟

git هي أداة تستخدمها فرق العمل بشكل عام و البرمجية منها بشكل خاص لإدارة إصدارات المشروع/العمل الذي يتعاون أعضاء الفريق على إنجازه.

و نعني بـ "إدارة الإصدارات" إمكانيّة تتبع التغييرات التي أجراها أعضاء الفريق على ملفات المشروع خلال مراحل تنفيذه المختلفة بحيث يمكن تحديد وقت و تاريخ كل تغيير إضافة لعضو الفريق المسؤول المتسبب بهذا التغيير.

## كيف نشأت git و لماذا؟

يعتبر لينوس تورفالدز (منشئ نظام لينكس) الأب الروحي لـ git، و تعود أسباب نشأة git إلى حاجة الفريق العامل على تطوير نواة لينكس حينها إلى أداة مناسبة لإدارة الإصدارات و صديقة لفرق عمل البرمجيات مفتوحة المصدر، خصوصاً بعد النزاع الذي نشأ بينهم و بين الشركة القائمة على تطوير نظام BitKeeper الذي كانوا يستعملونه وقتها (أبريل ٢٠٠٥) حيث أن الأخير مرخّص تجاريّاً و بالتالي لاستخدامه من قبل مجتمع كبير من المطوّرين تبعات قانونية كانت سبباً في إشعال فتيل النزاع.[1]

## لماذا تستخدم git؟

كما ذكرنا في التعريف، فإن الأهداف الرئيسية لاستخدام git تتلخّص في:

1. تسجيل التغييرات التي يجريها المطورون في شيفرات المشاريع البرمجية.
2. تسجيل المتسبّب بكلّ من تلك التغييرات و تاريخ كل منها.

[1] https://en.wikipedia.org/wiki/Git#History

3. إتاحة إمكانيّة التعاون في المشاريع البرمجية بحيث يمكن لأكثر من عضو في الفريق العمل على نفس الملف/الملفات في نفس الوقت دون ضياع أيّة مساهمات.

## تثبيت git:

تختلف آلية تثبيت git باختلاف نظام التشغيل المستخدم و فيما يلي التفاصيل:

- أنظمة ويندوز Windows: يتم التثبيت من خلال معالج تثبيت Installer يمكن تحميله من موقع الأداة الرسمي.
- أنظمة لينكس Linux: باستخدام سطر الأوامر، و اتباع أمر التثبيت المخصص لكل توزيعة من توزيعات لينكس و التي مكن مراجعتها من خلال موقع الأداة الرسمي.
- أنظمة ماك أو إس MacOs: باستخدام homebrew أو باستخدام معالج التثبيت الخاص بماك أو إس و الذي يمكن تحميله من موقع الأداة الرسمي.

بعد اتباع تعليمات التثبيت الخاصة بنظام التشغيل الذي تعمل عليه، يمكنك تجريب الأمر:

```
git --version
```

في سطر الأوامر للتأكد من تثبيت الأداة حيث يفترض أن يظهر رقم الإصدار الذي تم تثبيته للتوّ كما توضّح الصورة أدناه.[2]

```
muktar — -zsh — 37×7
/Users/muktar  > git --version
git version 2.35.1
/Users/muktar  >
```

---

[2] في متن هذا الكتاب سأستعمل سطر أوامر ماك أو إس MacOs نظراً لأنني أكتب هذا الكتاب على Macbook Pro، لكن بشكل عام يمكنك استخدام سطر الأوامر المتاح في نظام التشغيل الذي تستعمله بنفس الأسلوب.

## تعريف حساب git المحلّي

قبل أن نبدأ باستخدام git يفضّل أن نقوم بتعريف الحساب المحلّي الخاص بنا على git، و ذلك من خلال تنفيذ الأمرين:

```
git config --global user.name "Muktar SayedSaleh"

git config --global user.email "muktar@monjz.com"
```

بهذه الطريقة و عندما نقوم باستخدام git بعد قليل لحفظ أية تغييرات نقوم بها، فستظهر تلك التغييرات على أنّ المتسبب بها هو المستخدم صاحب الاسم و البريد الإلكتروني الذين أدخلناهما أعلاه.




ملحوظة أولى: لا تنسَ تغيير اسم المستخدم وعنوان البريد في المثال أعلاه إلى اسم المستخدم الخاص بك و عنوان بريدك الإلكتروني.

ملحوظة ثانية: كوننا استعملها الوسيط العام global-- فهذا يعني أن الحساب الذي قمنا بتعريفه سيكون هو الحساب المستخدم في جميع مستودعات git على هذا الحاسوب، ما لم نقم بتخصيص بعض المستودعات بغير ذلك صراحةً; و سنأتي على ذكر معنى المستودع بعد قليل.

## البدء باستخدام git

و الآن، يمكننا البدء باستخدام git لإدارة إصدارات المشروع الذي نعمل عليها حالياً، و ذلك من خلال الانتقال إلى مجلد المشروع، ثمّ إدخال الأمر التالي كما تبيّن الصورة أدناه، و ذلك لتحويله إلى مستودع git، حيث أنّنا نستخدم مصطلح "مستودع" Repository للإشارة إلى أي مشروع تدار إصداراته باستخدام git، و أمر التهيئة هو:

```
git init
```

```
[Desktop/git-book  > git init
Initialized empty Git repository in /Users/muktar/Desktop/git-book/.git/
Desktop/git-book - (master) >
```

و بهذا نكون قد قمنا للتوّ بإنشاء أوّل مستودع git بنجاح، مبارك!

## حالات الملفات في git

لقد أنشأنا مستودع git محليّاً في الفقرة السابقة، غير أنه فارغ و لا يحوي أيّة ملفّات، لذا سنقوم بإنشاء ملف جديد (و ليكن ملف html على سبيل التجربة)، ثم نقوم بحفظ ذلك الملف و إضافته إلى git.

فائدة: إذا لم تكن تعلم كيف تنشئ ملفات html و أحببت أن تتعلم فيمكنك الرجوع إلى كتابي المنشور مسبقاً و الذي يشرح ذلك بالتفصيل، أما بالنسبة لهدف هذا الكتاب فيمكنك إنشاء أي ملف نصّي عاديّ إذا أحببت.

على أيّة حال، سنستخدم لمثالنا في الشرح ملف HTML نضع به المحتوى التالي، و نحفظه باسم index.html

```
<html>

	<head><title>مرحباً بك في git!</title></head>

	<body><h1>مرحباً بك في git!</h1></body>

</html>
```

و الآن إذا عدنا إلى سطر الأوامر، و استعرضنا الملفات الموجودة في مجلّدنا الجديد فيجب أن نرى ملفّنا index.html فقط كما تبيّنه الصورة التالية:

```
git-book — -zsh — 86×5
Desktop/git-book - (master) > ls
index.html
Desktop/git-book - (master) >
```

ملحوظة: في لينكس و ماك أو إس، يستخدم الأمر ls لاستعراض الملفات الموجودة في المجلد، و يقابله الأمر dir في سطر أوامر ويندوز.

حسناً، الآن يمكننا أن نتحقق من حالة تتبع الملفات في git باستخدام الأمر

```
git status
```

و الذي سنستعمله كثيراً ضمن صفحات هذا الكتاب، حيث يستعمل هذا الأمر لرؤية الوضع الحالي لملفات المشروع بالنسبة لـ git، و في مثالنا الحالي بما أنّ الملف جديد فسيظهره لنا git على أنه ملف جديد لم يتم تتبعه بعد Untracked و لم نقم بالاحتفاظ بأي "إصدار" منه كما تبيّن لقطة الشاشة أدناه.

```
Desktop/git-book - (master) > git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html

nothing added to commit but untracked files present (use "git add" to track)
Desktop/git-book - (master) >
```

فبالنسبة لـ git تكون الملفات الموجودة في مستودع ما على إحدى حالتين:

- Tracked: و تشير إلى الملفات التي يعرفها git و المضافة إلى المستودع من قبل.
- Untracked: و تشير إلى الملفات الجديدة التي لم تُضف إلى المستودع بعد.

و سنتعلّم بعد قليل الأمر المستعمل لنقل الملفات من حالتها غير المتتبعة إلى حالة التتبّع مروراً بمرحلة الإدراج، و هي ما سنناقشه في الفقرة التالية.

## مرحلة الإدراج Staging

تمثل مرحلة الإدراج Staging الحالة التي تسبق مرحلة التتبّع مباشرة للملف الذي ننوي تتبّعه لأول مرّة، أو تتبّع التغييرات التي تمّت عليه لاحقاً، بما فيها حذفه، حيث لا يمكننا نقل الملف أو تغييراته من حالة عدم التتبع إلى حالة التتبع مباشرة دون المرور بمرحلة الإدراج.

و الملفات المُدرَجة هي ملفات جاهزة للإيداع كمساهمة commit في المستودع الذي تعمل عليه، وهو ما سنتعرض له بالتفصيل أثناء شرح عملية إيداع المساهمة commit في الفقرة اللاحقة، أما الآن و كوننا أتممنا العمل على الملف index.html، فلنقم بإضافته إلى مرحلة الإدراج باستخدام الأمر add بالشكل التالي:

```
git add index.html
```

و الآن إذا قمنا بتنفيذ أمر التحقق من حالة git (الأمر git status) فسنرى انتقال الملف من حالة عدم التتبع بشكل كلّي، إلى حالة الإدراج staging كما تبيّنه الصورة التالية:

```
git-book — -zsh — 87×11
Desktop/git-book - (master) > git add index.html
Desktop/git-book - (master) > git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   index.html

Desktop/git-book - (master) >
```

ملحوظة: يمكننا إضافة أكثر من ملف واحد إلى حالة الإدراج دفعةً واحدة من خلال الأمر git add --all حيث أنّ الوسيط --all يعني جميع الملفات. كما يمكننا إضافة مجلد كامل إلى حالة الإدراج من خلال كتابة اسم المجلد بدلاً من اسم الملف عند تنفيذ تعليمة add.

## إيداع المساهمات Committing

يعتمد git في إدارة الإصدارات على مفهوم "المساهمات" و المساهمة commit هي مجموعة من التعديلات التي تتم على الملفات و التي تحفظ ككتلة واحدة في شجرة تاريخ git، بحيث تعتبر كل مساهمة commit محطّة من محطّات سير المشروع و يتيح git العودة إلى أيّة محطّة مستقبلاً كما سنرى.

على أيّة حالة، لإيداع مساهمة يجب أولاً إدراج جميع التغييرات المطلوب إيداعها، ثمّ و بعد الإدراج Staging يتم الإيداع في المستودع من خلال الأمر commit مع مراعاة ترك رسالة مناسبة ترافق عملية الإيداع و تبين ما تم فيها بشكل مختصر عادة لسهولة المراجعة من قبل المساهِم نفسه أو زملائه في فريق العمل.

لإيداع العمل الذي قمنا به في الفقرة السابقة مع رسالة تفيد أنّه أول إيداع نقوم بتنفيذ الأمر التالي:

```
git commit -m "بسم الله الرحمن الرحيم، الإيداع الأول"
```

ينفذ الأمر commit عملية الإيداع، وتضيف "message" m- رسالة، و هكذا تكون بيئة الإدراج قد أودِعت في مستودعنا مع رسالة تقول "بسم الله الرحمن الرحيم، الإيداع الأول" كما يظهر في الصورة أدناه، و يلاحظ أنّ git يقوم تلقائياً بمنح ترقيم فريد لكل عمليّة إيداع، و يسمّى هذا الرقم "معرّف الإيداع" Commit ID.

```
git-book — -zsh — 87×11
Desktop/git-book - (master) > git commit -m "بسم الله الرحمن الرحيم، الإيداع الأول"
[master (root-commit) f1d5d3e] بسم الله الرحمن الرحيم، الإيداع الأول
 1 file changed, 8 insertions(+)
 create mode 100644 index.html
Desktop/git-book - (master) >
```

ملحوظة: بالنسبة لإيداع التغييرات على الملفات التي سبق و أن أودعت من قبل تنفيذ مرحلة الإدراج بشكل ضمني أثناء القيام بعملية الإيداع من خلال تمرير الوسيط a- بحيث تصبح التعليمة الكاملة كمايلي، مع نصيحتي بتحاشي استخدام هذه التعليمة لتفادي حالات إيداع ملفات غير مرغوب بإيداعها عن طريق الخطأ.

```
git commit -am "Your commit message here"
```

## إيداع نفس الملف مرّة أخرى!

حسناً، الآن بعد أن قمنا بإيداع الملف للمرّة الأولى، فلنقم بعمل أيّ تغيير ضمن الملف index.html (كإضافة سطر مثلاً أو تغيير محتوى الجملة النصيّة) و ذلك بهدف رؤية كيف سيقوم git بالتعرف على التغيير الحاصل عند تنفيذ الأمر git status كما توضحة الصورة أدناه،

و يرجى ملاحظة كلمة modified و التي تعني أن الملف مدرج سابقاً لكنه الآن خضع لتعديل جعل نسخته الحالية "معدّلة modified" بمعنى أنّها مختلفة عن آخر إدراج تم لها في git.

```
 1 file changed, 8 insertions(+)
 create mode 100644 index.html
[Desktop/git-book - (master) > git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
Desktop/git-book - (master) >
```

يمكننا الآن تكرار نفس خطوات عملية الإيداع السابقة، و إجراء إيداع ثانٍ لنفس الملف مع رسالة تفيد بسبب الإيداع كما تبينه الصورة التالية:

```
[Desktop/git-book - (master) > git add .
[Desktop/git-book - (master) > git commit -m "إضافة جملة جديدة إلى ملف الفهرس"
 [master 80bfe88] إضافة جملة جديدة إلى ملف الفهرس
 1 file changed, 1 insertion(+), 1 deletion(-)
Desktop/git-book - (master) >
```

## الاطلاع على سجل الإيداعات

الآن و كوننا قمنا بعمل إيداعين في مستودعنا، ربّما يكون الوقت مناسباً لتجربة أمر الاطلاع على الإيداعات التي تمت بالفعل في المستودع من خلال تنفيذ الأمر:

```
git log
```

و الذي يعرض معرّفات عمليات الإيداع السابقة و تاريخ كل منها، و الرسالة التي تبيّن سبب الإيداع كما أدخلها المستخدم الذي قام بعملية الإيداع كما يظهر في الصورة أدناه مع ملاحظة أنّ ترتيب ظهور عمليات الإيداع عكسي (من الأحدث للأقدم)

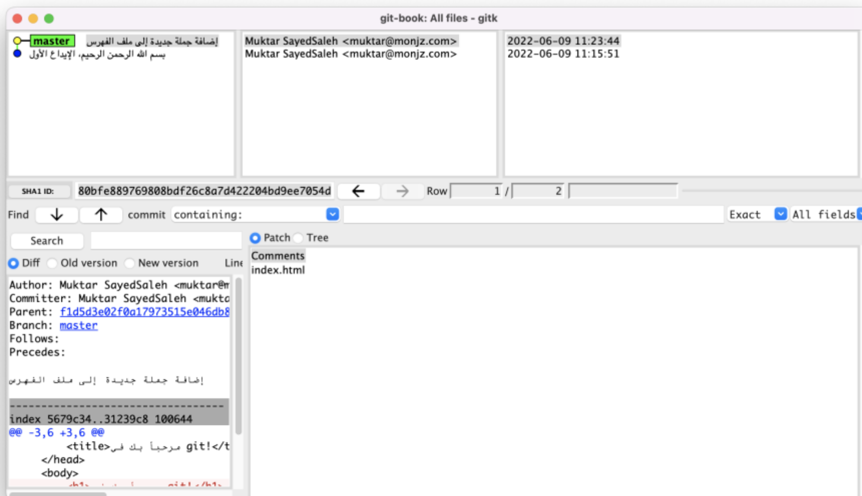
```
git-book — -zsh — 81×16
Desktop/git-book - (master) > git log
commit 80bfe889769808bdf26c8a7d422204bd9ee7054d (HEAD -> master)
Author: Muktar SayedSaleh <muktar@monjz.com>
Date:   Thu Jun 9 11:23:44 2022 +0300

    إضافة جملة جديدة إلى ملف الفهرس

commit f1d5d3e02f0a17973515e046db82a8918ab9bf56
Author: Muktar SayedSaleh <muktar@monjz.com>
Date:   Thu Jun 9 11:15:51 2022 +0300

    بسم الله الرحمن الرحيم، الإيداع الأول
Desktop/git-book - (master) >
```

فائدة: يمكن بشكل اختياري تثبيت أداة اسمها gitk تتيح لنا الاطلاع على السجلات من خلال واجهة استخدام مرئية GUI توضحها الصورة التالية مع تأكيدي أنّنا يجب أن نتعلّم git من خلال سطر الأوامر لضمان الفهم الكامل لآلية عملها و الاستفادة القصوى منها.



# التراجع عن عملية إيداع في git

يحدث أحياناً أن نقوم بعمليّة إيداع ما أو أكثر، ثمّ و لسبب أو لآخر نكتشف أننا بحاجة للتراجع عن عملية الإيداع تلك، تتيح git التراجع عن عمليّات الإيداع بأسلوبين اثنين أحدهما بسيط و سنناقشه في هذه الفقرة، و الآخر متقدم سنناقشه في الفصل الرابع من هذا الكتيّب بإذن الله تعالى.

أمّا الأمر البسيط فهو أمر git revert و الذي يقوم بالتراجع عن عملية إيداع معيّنة من خلال القيام بعملية إيداع أخرى معاكسة تماماً بحيث تكون النتيجة العودة بالملفات إلى حالتها التي كانت عليها قبل عملية الإيداع الأساسية التي نرغب بالتراجع عنها.

و لكي نفهم الأمر بشكل عملي، لنقم بالتراجع عن عملية الإيداع الثانية التي قمنا بها في الفقرة السابقة و التي قمنا بموجبها بإضافة بعض الكلمات إلى ملف html الذي أنشأناه، و للقيام بهذا الغرض نحتاج لمعرفة معرّف عملية الإيداع Commit ID المطلوب التراجع عنها و الذي يمكننا ببساطة نسخه ممّا يولده الأمر git log كما تظهر الصورة أدناه

```
git-book — -zsh — 81×16
Desktop/git-book - (master) > git log
commit 80bfe889769808bdf26c8a7d422204bd9ee7054d (HEAD -> master)
Author: Muktar SayedSaleh <muktar@monjz.com>
Date:   Thu Jun 9 11:23:44 2022 +0300

    إضافة جملة جديدة إلى ملف الفهرس

commit f1d5d3e02f0a17973515e046db82a8918ab9bf56
Author: Muktar SayedSaleh <muktar@monjz.com>
Date:   Thu Jun 9 11:15:51 2022 +0300

    بسم الله الرحمن الرحيم، الإيداع الأول
Desktop/git-book - (master) >
```

معرف عملية الإيداع

ثمّ تنفيذ الأمر التالي مع مراعاة استبدال <commit-id> بمعرّف عملية الإيداع :

```
git revert <commit-id>
```

```
commit 80bfe889769808bdf26c8a7d422204bd9ee7054d (HEAD -> master)
Author: Muktar SayedSaleh <muktar@monjz.com>
Date:   Thu Jun 9 11:23:44 2022 +0300

    إضافة جملة جديدة إلى ملف الفهرس

commit f1d5d3e02f0a17973515e046db82a8918ab9bf56
Author: Muktar SayedSaleh <muktar@monjz.com>
Date:   Thu Jun 9 11:15:51 2022 +0300

    بسم الله الرحمن الرحيم، الإيداع الأول
Desktop/git-book - (master) > git revert 80bfe889769808bdf26c8a7d422204bd9ee7054d

[master b43f4dd] Revert "إضافة جملة جديدة إلى ملف الفهرس"
 1 file changed, 1 insertion(+), 1 deletion(-)
Desktop/git-book - (master) >
```

فائدة: git ذكي كفاية لمعرفة عملية الإيداع المقصودة بمجرّد إدخال أوّل ٨ أحرف من معرّف العملية إذا كنت لا ترغب بنسخ كامل المعرّف. و هذا ينطبق على جميع الأوامر التي تستقبل commit ID و منها الأمر revert.

*الفصل الثانـي*

# التفريعات Branches

---

## ما هي التفريعات branches؟

يحدث في معظم الأحيان عند العمل ضمن فرق العمل أن يكون كل عضو أو مجموعة أعضاء من الفريق مسؤولين عن تطوير جزء مستقل و منفصل من المشروع، و لهذا الغرض قد يرغب هؤلاء بالعمل على نسخة شبه مستقلّة من المشروع بحيث يتاح لهم أخذ الوقت الكافي للتطوير دون التأثير على باقي أعضاء الفريق، و هذا تماماً ما يتم استخدام التفريعات Branches من أجله في git.

بمعنى آخر، تسمح الفروع للمطورين بالعمل على أجزاء مختلفة من المشروع دون التأثير على الفرع الرئيسي، و عند تمام العمل يُدمج الفرع الجديد مع المشروع الرئيسي، بل تستطيع التبديل بين الفروع و العمل على مشاريع مختلفة دون أن يؤثر كل منها على الآخر، فاستخدام أسلوب التفريع هنا طريقة سريعة و خفيفة لإجراء التعديلات على المشروع.

## إنشاء فرع جديد في git

لنضف الآن بعض المزايا الجديدة في صفحة index.html التي أنشأناها في الفصل السابق، و كوننا نرغب أن نقوم بهذا في مستودعنا المحلي دون التأثير على المشروع الرئيسي أو تعطيله سنقوم بإنشاء فرع جديد باستخدام الأمر git branch:

```
git branch hello-world-images
```

لقد أنشأنا الآن فرعاً جديداً branch باسم hello-world-again، و يمكننا أن نتحقق من وجوده من خلال تنفيذ الأمر git branch (دون تمرير اسم أي فرع) مرة أخرى

```
git-book — -zsh — 81×5
Desktop/git-book - (master) > git branch hello-world-images
Desktop/git-book - (master) > git branch
  hello-world-images
* master
Desktop/git-book - (master) >
```

كما نرى فيوجد الآن فرع جديد باسم hello–world–images، و فرع آخر اسمه master و بجانبه علامة نجمة * تعني أننا الآن في هذا الفرع (في الفرع master) و ربّما تتساءل الآن من أين جاء الفرع master و أنا أتعلّم التفريعات للتوّ؟ و الجواب أنّ git تنشئ الفرع master بشكل آلي في كل مستودع جديد.

فائدة: بعض توزيعات git تستخدم الاسم main بدلاً من master لتلافي الإيحاء التاريخي الذي يعنيه لفظ السيّد master في بعض الثقافات الغربية.

تنبيه: يجب الانتباه إلى الفرع الذي تتواجد فيه لحظة إنشاء فرع جديد، لأن الفرع الجديد سيكون في لحظة إنشائه نسخة مطابقة تماماً للفرع الذي تم إنشاؤه منه.

## الانتقال إلى فرع آخر في git

يُستخدم الأمر checkout للانتقال من الفرع الحالي إلى فرع آخر نمرّر اسمه مع الأمر، و لتجربة ذلك يمكننا الانتقال إلى الفرع الجديد الذي أنشأناه توّا كمايلي:

```
git checkout hello-world-images
```

```
git-book — -zsh — 81×5
Desktop/git-book - (master) > git checkout hello-world-images
Switched to branch 'hello-world-images'
Desktop/git-book - (hello-world-images) >
```

و هكذا نكون قد نقلنا مساحة العمل الحالية من الفرع الرئيسي إلى الفرع الجديد، و الآن لنفتح محرراً نصياً لنجري بعض التعديلات.

فائدة: يمكننا استخدام الخيار –b مع الأمر checkout لنطلب من git أن ينتقل إلى الفرع الجديد في حال كان موجوداً، وإلّا أن ينشئه ثم ينتقل إليه مباشرة.

في هذا المثال سنقوم بإضافة اسمها img_hello_world.jpg إلى مجلد العمل ثمّ نضمنها ضمن صفحة index.html بإضافة السطر التالي:

<img src="img_hello_world.jpg" alt="Hello World from Space" />

و بهذا نكون قد أجرينا تغييرات على ملف index.html و أضفنا الملف الجديد img_hello_world.png إلى مجلد العمل، و إذا نفذنا git status سنجد مايلي:

```
git-book — -zsh — 82×13
Desktop/git-book - (hello-world-images) > git status
On branch hello-world-images
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        img_hello_world.png

no changes added to commit (use "git add" and/or "git commit -a")
Desktop/git-book - (hello-world-images) >
```

و كما تلاحظ فالملف index.html يحوي بعض التعديلات (modified) غير أنّه لم يُدرج بعد Not staged، و أمّا الملف img_hello_world.jpg فحالته Untracked كونه يضاف للمرة الأولى و لم يودع من قبل، و للقيام بايداع كلا التغييرين سنحتاج إلى إضافة كلا الملفين إلى بيئة الإدراج لهذا الفرع:

```
git add --all
```

و للتذكير فقط استخدمنا all-- بدلاً من أسماء الملفات المستقلة لإدراج جميع الملفات بتعليمة واحدة، و الآن لو قمنا بالتحقق من حالة المستودع فسنرى ما توضحه الصورة التالية:

```
git-book — -zsh — 82×13
[Desktop/git-book - (hello-world-images) > git add --all
[Desktop/git-book - (hello-world-images) > git status
On branch hello-world-images
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   img_hello_world.png
        modified:   index.html

Desktop/git-book - (hello-world-images) >
```

تمرين: لماذا يظهر كلا الملفين باللون الأخضر الآن؟

حسناً، إذا كنّا جاهزين لإيداع هذه التعديلات يمكننا إيداعها باستخدام الأمر git commit الذي تعلمناه سابقاً مع ترك رسالة مناسبة توضح الهدف من عملية الإيداع:

```
git commit -m "Added image to index.html"
```

ما يهمنا الآن و نريد التركيز عليه هو وجود فرع branch جديد يختلف عن الفرع الرئيسي master و الاختلاف يمكن في عملية الإيداع الأخيرة، و لنرى الآن مدى سرعة وسهولة العمل مع الفروع المختلفة ومدى كفاءتها في العمل. نحن الآن في الفرع hello-world-images، وأضفنا صورة إلى ذلك الفرع، و إذا استعرضنا الملفات التي في المجلد الحالي سنجد الملفين index.html و img_hello_world.jpg، بمعنى أنّنا نستطيع الآن رؤية الملف الجديد img_hello_world.jpg، وكذلك نستطيع رؤية التغييرات الحادثة في الشيفرة إذا فتحنا ملف index.html، و هو ما نتوقعه تماماً. والآن، لنرى ما يحدث عند تغيير الفرع إلى master.

```
git checkout master
```

لم تعد الصورة الجديدة جزءاً من هذا الفرع، و لو استعرضنا الملفات الموجودة في المجلد الحالي مرة أخرى لرأينا الآن أن الصورة الجديدة غير موجودة و كذلك الشيفرة التي أضفناها إلى index.html و هذا يرينا سهولة العمل مع الفروع وكيف أنها تسمح بالعمل على أشياء مختلفة في نفس الوقت!

```
git-book — -zsh — 82x13
[Desktop/git-book - (master) > ls
 index.html
[Desktop/git-book - (master) > git checkout hello-world-images
 Switched to branch 'hello-world-images'
[Desktop/git-book - (hello-world-images) > ls
 img_hello_world.png     index.html
 Desktop/git-book - (hello-world-images) >
```

## فرع الإصلاحات الطارئة Hot fixes

لنتخيل الآن أننا لم ننه العمل على الفرع hello-world-images، لكنّنا و لسبب هام نريد إصلاح خطأ طارئ في الفرع الرئيسي master دون المساس بذلك الفرع الرئيسي ولا فرع hello-world-images بما أننا لم ننه العمل عليه بعد، في تلك الحالة يمكننا ببساطة إنشاء فرع جديد للتعامل مع هذا الإصلاح الطارئ و يمكننا تسميته hot-fix اصطلاحاً:

```
git checkout -b hot-fix
```

و نكون الآن قد أنشأنا فرعاً جديداً من الفرع الرئيسي و أجرينا تعديلات عليه، و نستطيع الآن إصلاح الخطأ دون التأثير على الفروع الأخرى، و كما تتوقع سيتم الأمر بإصلاح الأخطاء ثم إيداع هذا الإصلاح في الفرع الجديد، و لكن كيف نجلب هذا الإصلاح إلى الفرع الرئيسي؟ الجواب من خلال دمج الفرعين و هو ما سنناقشه في الفقرات التالية.

## دمج الفروع في Git

على فرض أن الخطأ الذي أصلحناه في الدرس السابق أصبح جاهزاً للدمج، يمكننا الآن أن نقوم بدمج الفرع hot-fix مع الفرع master باتباع الخطوتين التاليتين:

- الانتقال إلى الفرع الهدف (master) الذي نرغب بدمج الفرع الآخر معه من خلال تعليمة git checkout master.
- دمج الفرع الآخر hot-fix مع الفرع الهدف من خلال الأمر git merge hot-fix.

في حال كون كل شيء على ما يرام سيتم الدمج و ستظهر المساهمات التي قمنا بها في فرع الإصلاحات العاجلة ضمن الفرع الرئيسي كما تبين الصورة أدناه.

```
git-book — -zsh — 80×10
[Desktop/git-book - (hot-fix) > git checkout master
Switched to branch 'master'
[Desktop/git-book - (master) > git merge hot-fix
Updating b43f4dd..ef2be67
Fast-forward
 img_hello_world.png | 0
 index.html          | 2 +-
 2 files changed, 1 insertion(+), 1 deletion(-)
 create mode 100644 img_hello_world.png
[Desktop/git-book - (master) > git log
```

## حذف الفروع في git

في حال لم نعد بحاجة إلى فرع ما لأي سبب كان فيمكننا حذفه من خلال الأمر

```
git branch -D branch_name
```

حيث يمثل branch_name اسم الفرع المطلوب حذفه، و يراعى أن يكون الحرف الوسيط D بالحالة الكبيرة، كما تبين الصورة أدناه.

```
git-book — -zsh — 81×9
[Desktop/git-book - (master) > git branch -D hot-fix
Deleted branch hot-fix (was 052b7fe).
[Desktop/git-book - (master) > git branch
  hello-world-images
* master
Desktop/git-book - (master) >
```

# تعارض عمليات الدمج Merge conflicts

في الفقرة قبل السابقة، تمت عملية دمج الفرعين بشكل سلس و دون مشاكل و ذلك نظراً لعدم وجود أكثر من شخص يعملون على نفس الملفات في الوقت نفسه، لكن أحياناً و عند العمل ضمن فريق، لا تكون الأمور بهذه البساطة، و لا يستطيع git أحياناً معرفة الطريقة المثلى لدمج مساهمات المساهمين عندما يتصادف وجود تعديلين أو أكثر في نفس الملف و في نفس الأسطر، عندها يحدث ما يسمّى بتعارض الدمج Merge conflict و الذي يتطلب التدخل اليدوي من أحد المساهمين للقيام بعملية حلّ التعارض Conflict resolve بحيث يمكن إتمام الدمج بسلاسة.

و لشرح هذه الحالة لا بدّ من اصطناع حالة تستوجب التعارض أولاً، ثمّ سنرى معاً كيف يحدث التعارض، و الأهم كيف يمكن حلّه!

حسناً، لاختلاق حالة التعارض قم بعمل فرع جديد ثمّ عدّل السطر رقم 5 مثلاً من ملف index.html في نفس الفرع الحالي، و قم بحفظ المساهمة، ثمّ انتقل إلى الفرع الجديد و قم بتعديل مختلف في نفس السطر و في نفس الملف (السطر ٥ من الملف index.html) ثمّ قم بحفظ المساهمة.

الآن عندما تعود إلى الفرع الرئيسي و تحاول دمج الفرع الجديد سيظهر التعارض و سترى كلمة Conflict أمام الملفات المتعارضة كما تبيّنه الصورة التالية:

```
git-book — -zsh — 89×10
Desktop/git-book - (master) > git merge branch1
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
Desktop/git-book - (master) >
```

# حل التعارض Conflict Resolve

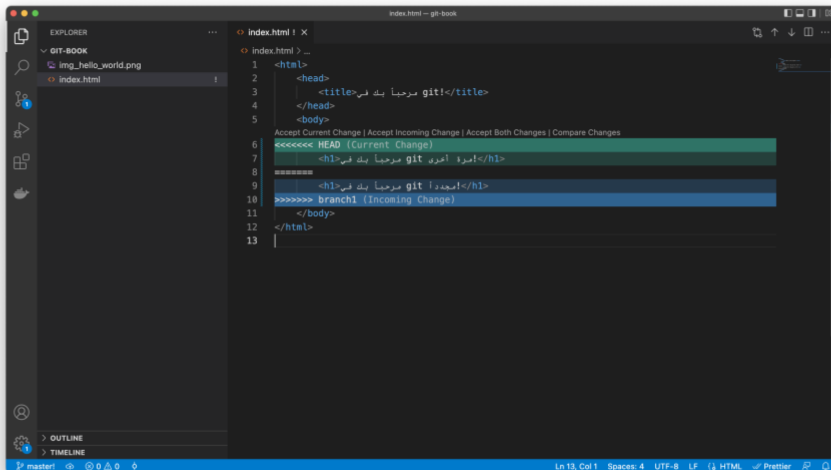
حسناً كما رأينا في الفقرة السابقة فقد حصل تعارض عندما حاولنا دمج الفرعين معاً نتيجة لحقيقة أننا قمنا بإجراء تعديلات مختلفة في نفس الأماكن من نفس الملفات و بالتالي لن يستطيع git تحديد طريقة لدمج المساهمات معاً بشكل آلي في الملفات المتأثرة بالتعارض، يمكننا من خلال الأمر status git معرفة الملفات التي تحوي تعارضات حيث سيظهرها git و بجانبها وصف both modified بمعنى أنّ الملف خضع لتعديل في كلا الفرعين، كما يظهر في الصورة أدناه.

```
git-book — -zsh — 89×10
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
	both modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
Desktop/git-book - (master) >
```

لو قمنا الآن بفتح الملف المتأثر باستخدام المحرر النصّي، سنجد أنّ git قام بوضع علامات تحدد مناطق التعارض ضمن الملف متوقعاً منا القيام بتعديلات يدوية على الملف لحل التعارضات، و من الجدير بالذكر أن بعض المحررات النصيّة المتقدمة تسهّل عمليّة حل التعارضات البسيطة من خلال بعض الأدوات كما في حالة VS Code مثلاً كما يظهر في الصورة أدناه.



يمكننا بوضوح رؤية أن git ترك التعديلين المتعارضين مع إشارة واضحة لاسم الفرع الذي جاء منه كل تعديل (head يعني الفرع الذي نحاول الدمج إليه، master في مثالنا أعلاه).

و الآن يمكننا إبقاء التعديل المطلوب و حذف التعديل غير المرغوب به ليعود الملف إلى حالة طبيعية، ثمّ نقوم بعد ذلك بعمل التعليمات git add للملفات التي تم تعديلها و حل التعارضات فيها، ثمّ التعليمة git merge –continue كما تظهره الصورة أدناه.

```
git-book — -zsh — 89×10

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
Desktop/git-book - (master) > git add index.html
Desktop/git-book - (master) > git merge --continue
[master c148406] Merge branch 'branch1'
Desktop/git-book - (master) >
```

بهذا نكون قد غطّينا الأوامر الأساسيّة في git للتعامل مع المستودع محليّاً (على الحاسب الخاص بنا)، و سنقوم في الفصل القادم بشرح كيفية التعامل مع مستودع بعيد مستعرضين منصة GitHub كمثال على منصّات git البعيدة، مع الإشارة إلى أنّ غالبيّة ما سنغطّيه عند الحديث عن GitHub ينطبق على جميع المنصّات الشهيرة مثل BitBucket أو GitLab.

الفصل الثالث

# مدخل إلى GitHub

---

## git أداة عمل موزّع Distributed

صممّت git في الأصل لتتيح لأعضاء فرق العمل المساهمة بشكل مستقلّ تماماً بحيث يملك كل عضو من أعضاء الفريق نسخته الخاصّة clone من المستودع الخاص بالمشروع الذي يتم العمل عليه، ثمّ عند الحاجة لدمج المساهمات القادمة من مساهمين مختلفين يعملون على أكثر من جهاز يمكن الاعتماد على مستودع بعيد Remote واحد أو أكثر يكون بمثابة المصدر الموثوق للشيفرة اصطلاحاً Source of truth مع أنه في نهاية المطاف ما هو إلّا نسخة Clone من المستودع نفسه.

لتنظيم هذه العملية و تسهيلها ظهرت مجموعة من المنصّات أشهرها GitHub و BitBucket و GitLab و سنقوم في هذا الفصل بالحديث عن GitHub بشكل مفصّل.

| |
|---|
| تنبيه: كثيراً ما يستعمل المبتدئون GitHub للإشارة إلى git أو العكس، مع أنّهما شيئآن مختلفان تماماً تماماً كما سبق و أوضحنا، لذلك أرجو توخّي الدقّة عند استعمال المصطلحات. |

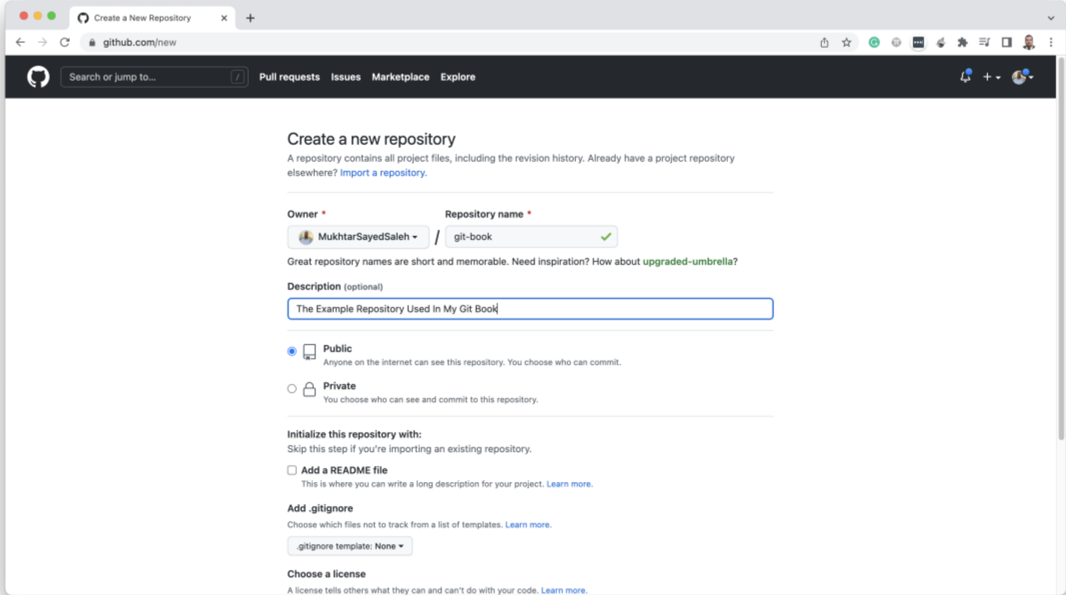
## تسجيل حساب جديد في GitHub

تسجيل الحساب في GitHub بسيط و مباشر و كل ما تحتاجه هو الانتقال إلى الموقع github.com ثمّ الضغط على زر sign up ثمّ تقوم بملء النموذج و تسجيل الحساب بشكل تقليدي، و ربّما تكون التوصية الوحيدة التي أنصح بها هنا هي تسجيل الحساب بنفس عنوان البريد الذي استخدمناه في تهيئة git في الفصل الأوّل.



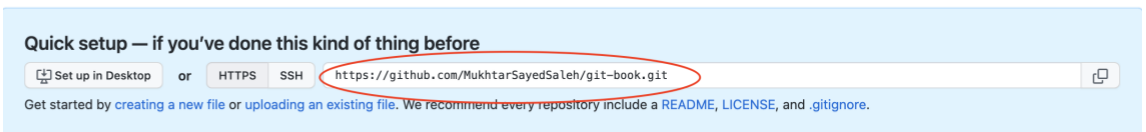
# إنشاء مستودع على GitHub

بعد تسجيل الحساب يمكننا إنشاء مستودع على GitHub مباشرة من خلال اختيار Create new repository و ملأ تفاصيل المستودع (اسمه و وصفه .. إلخ) كما تبين الصورة أدناه.




أهمّ خيار ربّما في هذه الشاشة هو خيار جعل المستودع خاصّاً Private، أو عامّاً Public، حيث أنّ جعل المستودع عاماً Public يعني أنّه أصبح متاحاً لكل الناس للاطلاع عليه، و نسخه، و المساهمة فيه حتّى إذا لم تضبط بعض إعدادات الحماية.

الآن نضغط على زر Create repository لإتمام إنشاء المستودع البعيد على GitHub و خلال زمن لا يذكر سننتقل آلياً إلى صفحة المستودع الذي قمنا بإنشائه، و لعلّ أهمّ ما في الصفحة في هذه المرحلة هو رابط المستودع البعيد الذي يظهر في الصورة أدناه و الذي سنستعمله بعد قليل لمزامنة مستودعنا المحلّي الذي أعددناه سابقاً، مع المستودع البعيد الذي أنشأناه للتوّ.



## تعريف المستودع البعيد محليّاً

لمزامنة المستودع المحلّي الذي أعددناه في فقرة سابقة على حاسوبنا الشخصي، مع المستودع البعيد الذي أنشأناه للتوّ على GitHub نحتاج إلى تعريفه ضمن المستودعات البعيدة محليّاً، و يتم ذلك من خلال الأمر git remote كما يلي:

```
git remote add <remote-name> <github-repo-url>
```

حيث يعبّر remote-name عن الاسم الذي سنسمّي به المستودع البعيد (عادة يسمّى origin)، بينما يعبّر github-repo-url عن رابط المستودع البعيد الذي رأيناه في الفقرة السابقة، و لإضافة مستودعنا البعيد إلى المحلّي يصبح الأمر:

```
git remote add origin https://github.com/MukhtarSayedSaleh/git-book.git
```

و الآن يمكننا استعراض المستودعات البعيدة من خلال الأمر git remote -v للتأكد من أنّ المستودع أضيف بشكل صحيح كما يظهر في الصورة أدناه.

```
git-book — -zsh — 89×10
[Desktop/git-book - (master) > git remote add origin https://github.com/MukhtarSayedSaleh/]
git-book.git
[Desktop/git-book - (master) > git remote -v                                              ]
origin  https://github.com/MukhtarSayedSaleh/git-book.git (fetch)
origin  https://github.com/MukhtarSayedSaleh/git-book.git (push)
Desktop/git-book - (master) >
```

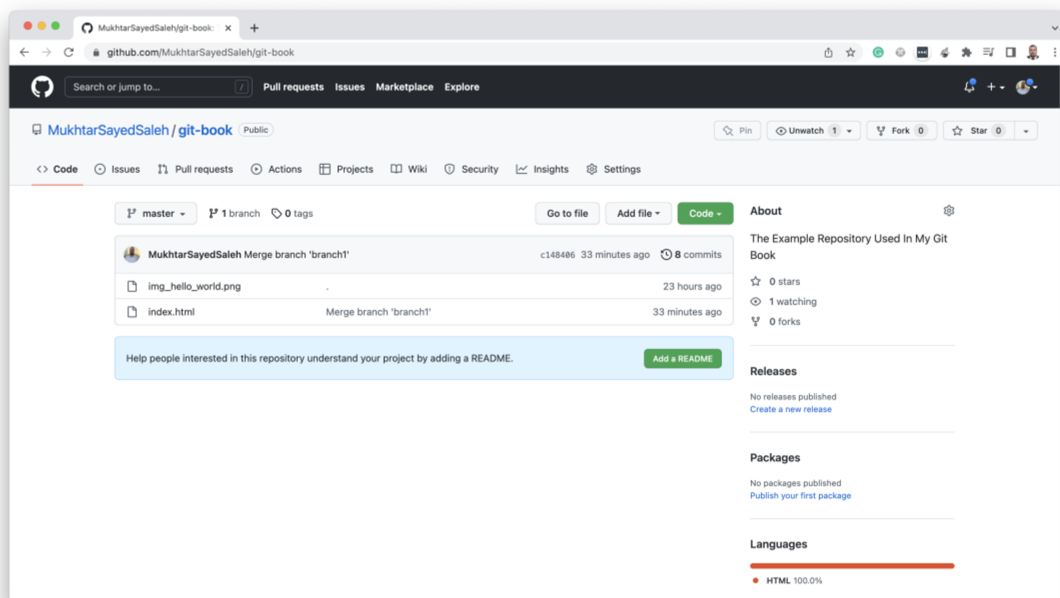
## إرسال تعديلاتنا المحليّة إلى المستودع البعيد Push

يصطلح في git على استخدام الفعل دفع push للتعبير عن إرسال التعديلات المحليّة إلى المستودع البعيد، و كثيراً ما يقال "تدفع التعديلات" بمعنى أنّ نزامنها مع المستودع البعيد من خلال الأمر push.

الآن مستودعنا البعيد origin و الموجود على GitHub لا يحوي أيّة ملفات كونه أنشئ للتوّ، لذلك يمكننا مزامنة التعديلات الموجودة محليّاً معه من خلال تنفيذ الأمر git push origin (بمعنى أنّنا سندفع التعديلات المحليّة إلى المستودع البعيد ذي الاسم origin) كما في الصورة أدناه.

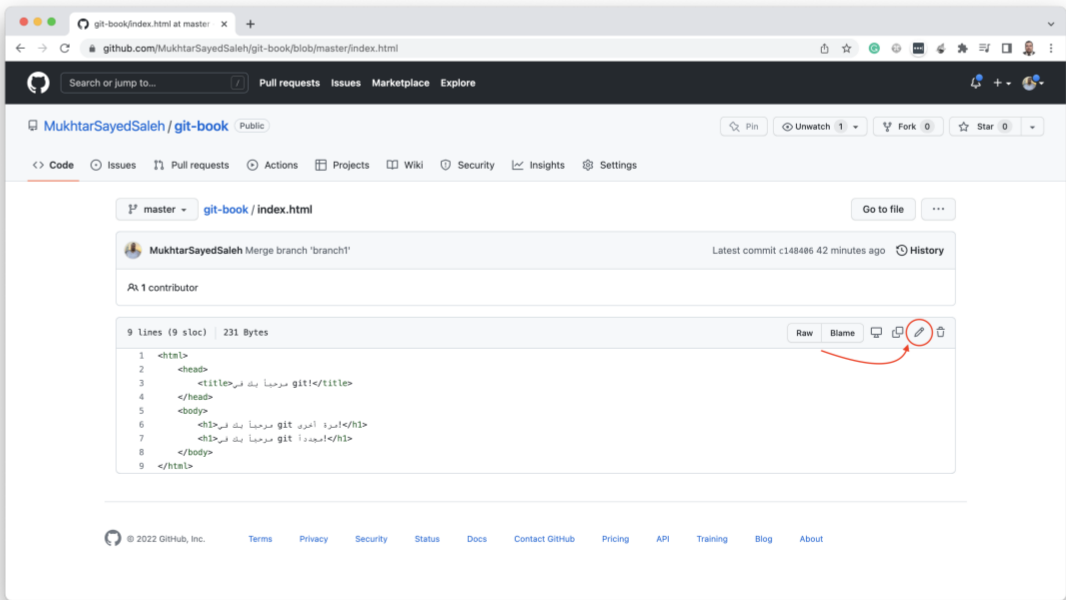
```
git-book — -zsh — 89×12
[Desktop/git-book - (master) > git push --set-upstream origin master
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 16 threads
Compressing objects: 100% (20/20), done.
Writing objects: 100% (23/23), 2.13 KiB | 1.07 MiB/s, done.
Total 23 (delta 8), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (8/8), done.
To https://github.com/MukhtarSayedSaleh/git-book.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
Desktop/git-book - (master) >
```

و بمجرّد تنفيذ الأمر ستظهر الملفات على GitHub كما تظهر لقطة الشاشة أدناه.



# تعديل الملفات في GitHub مباشرة

يتيح GitHub لمن يرغب واجهة ويب لتعديل الملفات مباشرة ضمن الموقع و دون الحاجة لمغادرته، و بحيث يتم حفظ المساهمات (التعديلات) التي تتم ضمن GitHub كمساهمة مباشرة على المستودع البعيد، و لتجربة ذلك نضغط على أحد الملفات ثم نختار أمر التعديل الظاهر في الصورة أدناه.




ثمّ نقوم بإجراء بعض التعديلات و نختار الحفظ، و سيطلب منا GitHub إدخال رسالة المساهمة المناسبة (Commit message) و حفظ التعديل كمساهمة جديدة Commit و كأنّنا قمنا بذلك من خلال الأمرين git add و git commit كما نفعل عادة!

Commit changes

Add an optional extended description...

muktar@monjz.com

Choose which email address to associate with this commit

Commit directly to the master branch.

Create a new branch for this commit and start a pull request. Learn more about pull requests.

Commit changes Cancel

## جلب تعديلات المستودع البعيد إلى المستودع المحلّي Pull

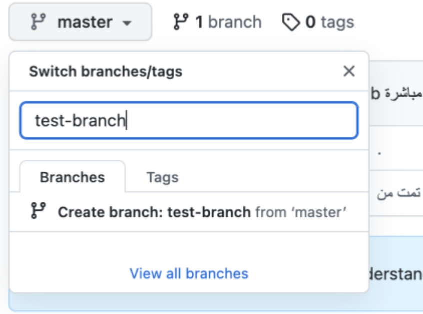
يصطلح في git على استخدام الفعل سحب pull للتعبير عن طلب التعديلات البعيدة و جلبها إلى المستودع المحلّي، و كثيراً ما يقال "نسحب التعديلات" بمعنى أنّ نزامن المستودع المحلّي مع التحديثات التي دفعت على المستودع البعيد من قبل زملاء العمل، و لتجربة ذلك فلنقم بجلب التعديل الذي أجريناه على المستودع البعيد (مستودع GitHub) في الفقرة السابقة إلى المستودع المحلّي و ذلك من خلال تنفيذ الأمر git pull كما توضّح الصورة أدناه.

```
git-book — -zsh — 89×13
Desktop/git-book - (master) > git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 798 bytes | 266.00 KiB/s, done.
From https://github.com/MukhtarSayedSaleh/git-book
   c148406..c4d98de  master     -> origin/master
Updating c148406..c4d98de
Fast-forward
 index.html | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
Desktop/git-book - (master) >
```

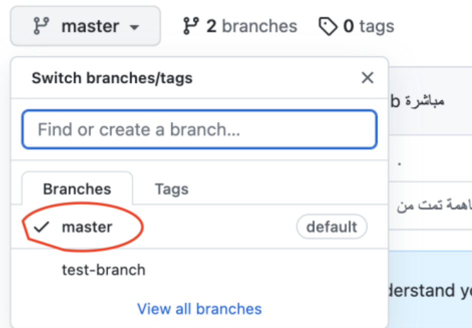
ملحوظة: أحياناً، و إذا كان ثمّة تعديلات على المستودع البعيد لم يتم سحبها بعد، فلن يسمح لك git بالقيام بعمل دفع push و سيطلب منك سحبها أولاً، و قد تحدث أثناء ذلك تعارضات تتطلّب الحل بنفس الأسلوب الذي استعرضناه في الفصل الماضي.

## إنشاء الفروع Branches في منصة GitHub

نستطيع إنشاء فرع جديد للمشروع الذي نعمل عليه في منصة GitHub بالدخول إلى المستودع البعيد الخاص بنا و النقر على زر الفرع الرئيسي master، ثمّ كتابة اسم الفرع الجديد مع الحرص على كتابة اسم يصف الفرع و الهدف منه، ثم ننقر على زر Create Branch:




ينبغي للفرع test−branch أن يكون قد أُنشئ الآن و صار نشطاً، و نستطيع معرفة أي الفروع نحن فيها بالنظر إلى زر الفرع كما في الصورة أدناه، حيث نرى أنه يذكر أننا في الفرع master




## مخطط عمل جيت هب GitHub Workflow

لأنّ git مرنة بحيث لا تفرض على مستخدمها أية قيود في ما يتعلق بكيفية استخدامها و توظيفها ضمن فريق العمل، قامت بعض الشركات بعمل معايير standards لاستخدام git، و لعلّ أشهرها هو مخططات عمل GitHub أو GitHub Workflow و الذي يركّز على توظيف إمكانيّات git و GitHub كافة للعمل بشكل معياري في فرق العمل بحيث لا يحتاج المساهمون الجدد أو الموظفون الجدد إلى كثير من الشرح للبدء بالمساهمة الفعلية في المشاريع.

يركز أسلوب العمل هذا على الاستخدام المكثّف للفروع بحيث يمكّن الفرق من اختبار التعديلات و المزايا الجديدة بحرية و نشر البرمجيات deployment بانتظام، و فكرة عمل هذا المخطط كما يلي:

1) إنشاء فرع جديد: ينشئ عضو الفريق فرعاً جديداً بهدف عدم التأثير على الفروع الأخرى المستقرّة، عادة تكون الفروع المستقرّة بأسماء master الذي يستعمل للبيئة الإنتاجية production و develop الذي يستعمل لبيئات التطوير، أو اختبارات ضمان الجودة QA أو تجارب العملاء المبكّرة staging.

2) إيداع المساهمات: يقوم عضو الفريق بإيداع المساهمات الخاصة بمهمته ضمن الفرع الذي أنشأه في الخطوة السابقة.

3) فتح طلب سحب Pull request: في GitHub طلبات السحب Pull Request هي عملية دمج معلّقة تتطلّب مراجعة المساهمات من قبل واحد أو أكثر من أعضاء الفريق و الموافقة عليها قبل أن يتم دمجها، و في بعض البيئات تسمّى طلب دمج Merge Request – و المقصود بها هنا أنّ عضو الفريق انتهى من المساهمات في فرعه و الآن يطلب من فريقه الموافقة على دمجها مع فروع المشروع المستقرة بهدف نشرها.

4) مرحلة المراجعة Code Review: تابعة للمرحلة السابقة حيث يقوم أعضاء الفريق الآخرون (أو قادته) بمراجعة المساهمات التي طلب زميلهم دمجها، و عادة ما تحدث نقاشات مفيدة و إيجابية هنا لتحسين جودة المساهمات.

5) مرحلة الدمج و النشر: يتم دمج الفرع المطلوب و نشره Deploy بشكل آلي أو يدوي.

# خدمة GitHub Pages لاستضافة صفحات الويب

تتيح منصة GitHub خدمة استضافة صفحات الويب الموجودة في مستودع بعيد مباشرة على GitHub و ذلك من خلال إنشاء مستودع عام جديد و تسميته باسم username.github.io حيث أن username هو اسم المستخدم الخاص بك.

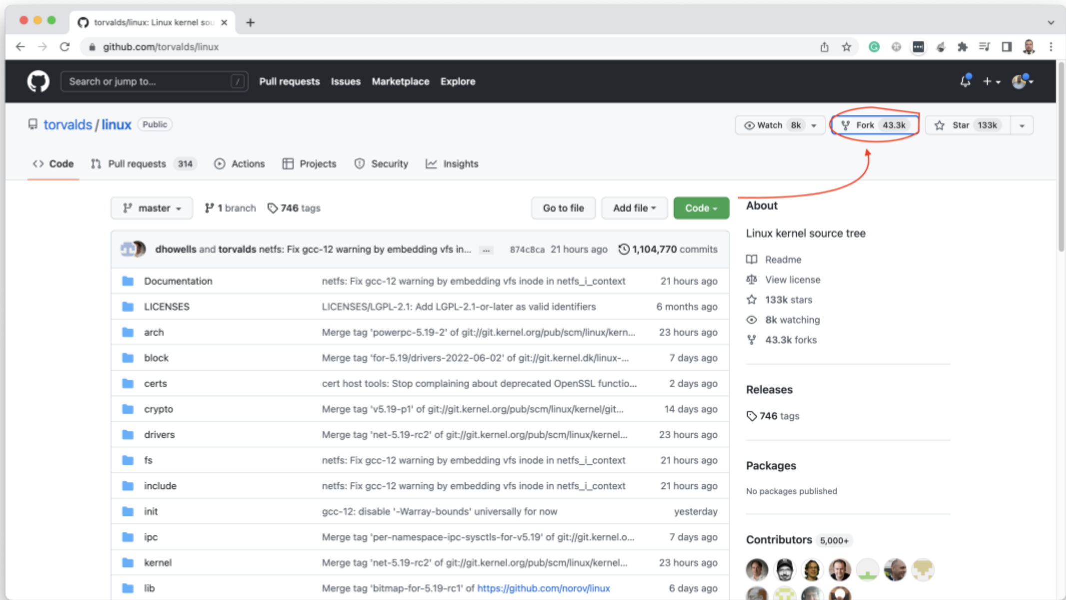
بهذه الطريقة و بمجرّد دفعك لأيّة صفحات ويب إلى هذا المستودع البعيد ستكون هذه الصفحات متاحة للعموم و كأنّها مستضافة على web server و برابط مباشر من GitHub Pages يمكنك العثور عليه من تبويب صفحات جيت هب في إعدادات المشروع.

## الاشتقاق في GitHub

تتيح منصة GitHub لمستخدميها اشتقاق مستودعات جديدة خاصة أو عامة من مستودعات المستخدمين الآخرين العامّة، و ذلك بواسطة الأمر fork الذي يسمح لك بالبدء بمستودع جديد بناءً على ما تم بالفعل في مستودع مشروع آخر سواء كان المشروع الآخر مملوكاً لك أو عامّاً.

على أن fork نفسها ليست أحد الأوامر في نظام git، بل هي خاصية توفرها منصة GitHub و المنصات المشابهة

لاشتقاق مستودع ما يمكنك بكل بساطة الدخول إلى المستودع المطلوب ثمّ الضغط على زر Fork الذي توضّحه الصورة التالية.

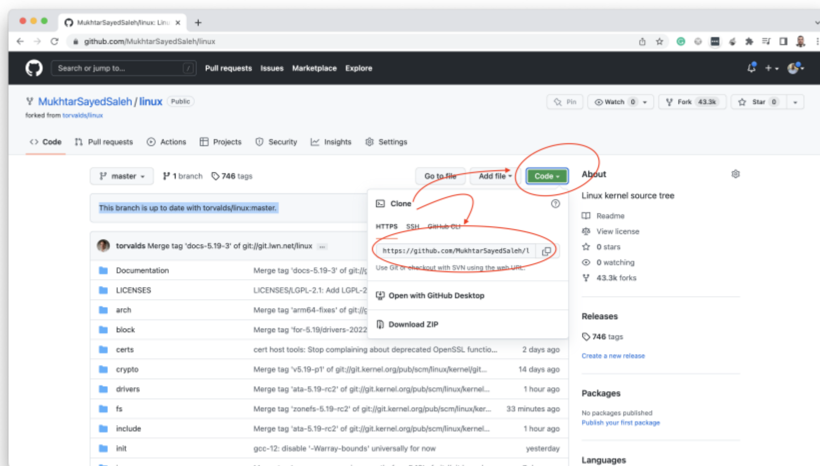



## استنساخ المستودعات Repositories Clonning في GitHub

لقد قمنا في الفقرة السابقة بإنشاء اشتقاقنا الخاص من مستودع Linux مفتوح المصدر في GitHub، غير أن ذلك المستودع لا يوجد إلا على منصة GitHub حالياً، و من البديهي أنّنا نريد حالياً مواصلة العمل عليه في مستودع git المحلي كون المحرّر النصّي الموجود في GitHub

مناسب فقط للتعديلات و المساهمات البسيطة جداً و ليس للتطوير الحقيقي، و لهذا تتيح GitHub و غيرها من المنصّات المشابهة ميزة الاستنساخ Clonning التي تتيح استنساخ مستودع بعيد إلى الحاسب المحلي، و حيث نقول استنساخ فنحن نعني ذلك لأن المستودع الذي سينشأ محليّاً سيحوي كامل المساهمات و الأفرع التي تمت ضمن المستودع الأصلي.

يتم الاستنساخ من خلال الأمر git clone ثم تمرير رابط المستودع المراد استنساخه، و للحصول على الرابط يمكننا الضغط على زر code في GitHub.




```
Desktop — -zsh — 80×24
muktar/Desktop  > git clone git@github.com:MukhtarSayedSaleh/linux.git
Cloning into 'linux'...
remote: Enumerating objects: 8841294, done.
remote: Counting objects: 100% (279/279), done.
remote: Compressing objects: 100% (159/159), done.
remote: Total 8841294 (delta 194), reused 147 (delta 120), pack-reused 8841015
Receiving objects: 100% (8841294/8841294), 3.63 GiB | 14.35 MiB/s, done.
Resolving deltas: 100% (7338361/7338361), done.
Updating files: 100% (76960/76960), done.
warning: the following paths have collided (e.g. case-sensitive paths
on a case-insensitive filesystem) and only one from the same
colliding group is in the working tree:

  'include/uapi/linux/netfilter/xt_CONNMARK.h'
  'include/uapi/linux/netfilter/xt_connmark.h'
  'include/uapi/linux/netfilter/xt_DSCP.h'
  'include/uapi/linux/netfilter/xt_dscp.h'
  'include/uapi/linux/netfilter/xt_MARK.h'
  'include/uapi/linux/netfilter/xt_mark.h'
  'include/uapi/linux/netfilter/xt_RATEEST.h'
  'include/uapi/linux/netfilter/xt_rateest.h'
  'include/uapi/linux/netfilter/xt_TCPMSS.h'
  'include/uapi/linux/netfilter/xt_tcpmss.h'
  'include/uapi/linux/netfilter_ipv4/ipt_ECN.h'
```

**تنبيه هام:** قبل الاستنساخ تأكّد من قراءة ملف الترخيص License file فكثيراً ما يلجأ المطورون (المبرمجون) لاستعمال مكتبات، أو أجزاء من مكتبات، أو حتى أسطر محددة من ملفات، سبق و أن نشرت تحت إحدى رخص البرمجيات مفتوحة المصدر دون انتباه لطبيعة الترخيص المصاحب لأصل الشيفرة معادة الاستخدام و انعكاسات ذلك على المنتج الذي يعمل عليه المطوّر، حيث أن ذلك قد يحمّلك عواقب و يضعك تحت التزامات قانونية، نذكر منها:

١ –نشر منتجك النهائي تحت نفس الرخصة: قد يعني استخدامك لشيفرة تحت رخصة معينة أنك مجبر –بالقانون– على نشر منتجك النهائي عندما يجهز بشكل مفتوح المصدر و تحت نفس الرخصة الأصلية.

٢ –منع الاستخدام التجاري: قد يعني استخدامك لشيفرة تحت رخصة معينة أنك مجبر –بالقانون– ألا تستخدم منتجك النهائي بشكل تجاري أو ربحي إطلاقاً!

٣ –منع الاشتقاق: بمعنى السماح للآخرين باستعمال هذا المنتج "كما هو" دون إتاحة البناء عليه أو تعديله أو ترجمته ... إلخ، و قد يكون لهذا الأمر أثر رجعي على منتجك أنت أيضاً.

٤ –إلزامية النسبة: بحيث تكون مجبراً بالقانون على ذكر اسم الشخص أو الكيان مالك الأجزاء المقتبسة أو المكتبات المستعملة.

لذلك في المرة القادمة و قبل أن تقوم ب"نسخ–لصق" لأي شيفرة برمجية تجدها على الانترنت، تأكد من قراءة الترخيص بشكل جيد لما لذلك من التزامات قانونية هامّة قد يجهلها الكثيرون.

*الفصل الرابع*

# مواضيع متقدمة في git

## ملف التجاهل gitignore.

لاشك أنه ستكون أجزاء أو ملفات من المشروع الذي تعمل عليه لا تريد مشاركتها مع غيرك ممن يطلع على شيفرة المشروع، و من أمثلة تلك الملفات:

- ملفات السجلات log files.
- الملفات المؤقتة temporary files.
- الملفات المخفية.
- الملفات الشخصية.
- ملفات الكلمات السريّة و مفاتيح المصادقة.
- غير ذلك من الملفّات ذات الحساسيّة.

تستطيع أن تطلب من git أن يتجاهل تلك الملفات أو الأجزاء من المشروع التي ينبغي له تجاهلها باستخدام ملف gitignore. (لاحظ أن اسم الملف يبدأ بنقطة و بالتالي سيكون مخفياً بشكل افتراضي في أنظمة لينكس و ماك أو إس) و لن يتعقب git الملفات و المجلدات المحددة في gitignore. رغم أنه يتتبع ملف gitignore. نفسه.

يجب أن يتم إنشاء الملف gitignore. في أعلى مستوى من مجلدات المشروع (في المجلد الجذر root بتعبير آخر)، ثم و بعد إنشاء الملف نستخدم أيّ محرر نصي، و نضيف القواعد التي تحدد الملفات التي نرغب بتجاهلها، و حين نقول قواعد فنحن نعني ذلك لأنّنا قد نكون في حالات نحتاج فيها لتجاهل الكثير من الملفات دفعة واحدة كتلك التي لها امتداد محدد مثلاً أو التي تطابق أسماؤها نمطاً معيّناً ... إلخ، و فيمايلي القواعد التي نستطيع تعريفها:

| النمط | الشرح – المطابقات | أمثلة |
|---|---|---|
| | ليس للأسطر الفارغة أي تأثير في ملف gitignore. | |

| | | |
|---|---|---|
| # text comment | ليس للأسطر التي تبدأ بإشارة # أي تأثير في ملف gitignore. و إنما تستخدم لترك تعليقات لتوضيح شيء ما من قبل المطوّرين عادة | |
| name | جميع ملفات name و مجلدات name، و الملفات و المجلدات التي في أي مجلد name | /name.log<br>/name/file.txt<br>/lib/name.log |
| name/ | الانتهاء بالشرطة المائلة slash يشير إلى أن المطلوب مطابقته هو المجلد المذكور، و يطابق جميع الملفات و المجلدات في أي مجلد name | /name/file.txt<br>/name/log/name.log<br>عدم وجود تطابق:<br>/name.log |
| name.file | جميع الملفات التي فيها name.file | /name.file<br>/lib/name.file |
| /name.file | الابتداء بالشرطة المائلة / يشير إلى أن النمط يطابق الملفات الموجودة في المجلد الجذر فقط، و هو مجلد root | /name.file<br>عدم وجود تطابق:<br>/lib/name.file |

| | | |
|---|---|---|
| lib/name.file | يحدد النمط الملفات في مجلدات بعينها، لتكون منسوبة دوماً إلى المجلد الجذر حتى لو لم نبدأ بالشرطة الدالة على المجلد الجذر، و هي الشرطة المائلة / | /lib/name.file<br>عدم وجود تطابق:<br>name.file<br>/test/lib/name.file |
| **/lib/name.file | الابتداء بعلامتي النجمة ** قبل الشرطة المائلة / يحدد أن النمط يطابق أي مجلد في المستودع، و ليس المجلد الجذر فقط | /lib/name.file<br>/test/lib/name.file |
| **/name | جميع مجلدات name و أي ملفات أو مجلدات في أي مجلد name | /name/log.file<br>/lib/name/log.file<br>/name/lib/log.file |
| /lib/**/name | جميع مجلدات name و الملفات و المجلدات في أي مجلد name داخل مجلد lib | /lib/name/log.file<br>/lib/test/name/log.file<br>/lib/test/ver1/name/log.file<br>عدم وجود تطابق:<br>/name/log.file |
| *file. | جميع الملفات ذات الامتداد file. | /name.file<br>/lib/name.file |

| | | |
|---|---|---|
| /name*/ | جميع المجلدات التي تنتهي بـ name | /lastname/log.file<br>/firstname/log.file |
| name?.file | تطابق العلامة ? محرفاً وحيداً غير محدد. | /names.file<br>/name1.file<br>عدم وجود تطابق:<br>/names1.file |
| name[a-z].file | تطابق محرفاً وحيداً في نطاق محدد، و يكون المحرف في تلك الحالة حرفاً من a حتى z، أو رقماً | /names.file<br>/nameb.file<br>عدم وجود تطابق:<br>/name1.file |
| name[abc].file | يطابق محرفاً واحداً في مجموعة محددة من المحارف، و في تلك الحالة يكون المحرف أحد الحروف الثلاثة a أو b أو c | /namea.file<br>/nameb.file<br>عدم وجود تطابق:<br>/names.file |
| name[!abc].file | يطابق محرفاً واحداً باستثناء المحارف المحددة في مجموعة، و هي a-b-c في تلك الحالة | /names.file<br>/namex.file<br>عدم وجود تطابق:<br>/namesb.file |
| *.file | جميع الملفات ذات الامتداد file. | /name.file |

| | | |
|---|---|---|
| | | /lib/name.file |
| name/<br>!name/secret.log | تحدد العلامة ! نفياً أو استثناءً، و تطابق جميع الملفات و المجلدات في أي مجلد name باستثناء name/secret.log | /name/file.txt<br>/name/log/name.log<br>عدم وجود تطابق:<br>/name/secret.log |
| *.file<br>!name.file | تحدد العلامة ! نفياً أو استثناءً، و تطابق جميع الملفات ذات الامتداد .file باستثناء name.file | /log.file<br>/lastname.file<br>عدم وجود تطابق:<br>/name.file |
| *.file<br>!name/*.file<br>junk.* | إضافة أنماط جديدة بعد النفي سيعيد تجاهل ملف تم تجاهله من قبل. جميع الملفات ذات الامتداد .file باستثناء تلك التي في مجلد name، إلا إذا كان اسم الملف junk | log.file<br>/name/log.file<br>عدم وجود تطابق:<br>/name/junk.file |

فائدة: يوجد موقع لطيف جداً اسمه https://gitignore.io يعطي قوالب جاهزة لملفات .gitignore مناسبة لكل لغة برمجة أو إطار عمل شهير لتسريع إعداد ما يجب تجاهله. جرّبوه!

# السفر في الزمن مع git checkout

نستطيع باستخدام git أن نعيد عقارب الساعة إلى الوراء مجازياً، بمعنى أنّ نعيد الملفات الموجودة في المستودع إلى الحالة التي كانت فيها لحظة مساهمة ما تماماً، و ذلك من خلال الأمر git checkout ثمّ ذكر معرّف المساهمة المطلوب العودة إليها.

```
git checkout <commit-id>
```

# الأمر Reset

ذكرنا في الفصل الأوّل أنّه توجد طريقة ثانية متقدّمة للتراجع عن مساهمة أو أكثر قمنا بها، و قلنا أنّنا سنغطيها في الفصل المتقدّم، و الآن قد حان الوقت للحديث عنها، ألا و هي طريقة الأمر reset.

يُستخدم الأمر reset عند الرغبة في إرجاع المستودع في التاريخ إلى مساهمة سابقة و له ثلاثة أنماط:

1. النمط الخشن hard--: و هو أخطر الأنماط حيث أنّه يعيد المستودع إلى مساهمة سابقة مع إلغاء جميع المساهمات و التغييرات التي حدثت بعد تلك المساهمة.

```
git reset --hard <commit-id>
```

2. النمط الناعم soft--: و هو شبيه بالسابق إلا أنّه لا يلغي المساهمات التالية، بل يبقيها و كأنّها تمت و تم عمل add لها إنّما لم يتم عمل commit (و لهذا اسمه سوفت).

```
git reset --soft <commit-id>
```

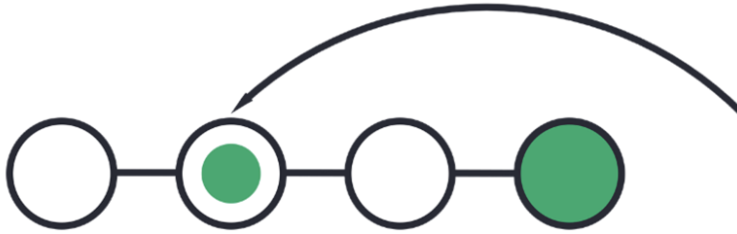
3. النمط الوسطي mixed--: شبيه بالنمط الناعم مع فارق أنّه يبقي التغييرات و كأنّها حدثت للتوّ و لم تسجّل في تاريخ المستودع بعد، و بالتالي إذا أردت الاحتفاظ بها فيجب أن تقوم بعمل git add ثم git commit.

```
git reset --mixed <commit-id>
```

اعتراف: شخصياً فكثيراً ما أستعمل هذا النمط للتراجع عن بعض المساهمات التي أضطر للقيام بها على سبيل التجربة أو لأسباب أخرى قبل الانتهاء الفعلي من العمل على ميزة ما.

على أيّة حال، و بغضّ النظر عن النمط الذي تريد استخدامه فيجب أوّلاً أن تقوم بتحديد المساهمة التي ترغب في العودة إليها (من خلال تحديد معرّفها Commit ID من خلال الأمر git log أو gitk).

يوضّح الشكلان التخيّليان التاليان ماذا يحدث عند القيام بالأمر git reset --hard حيث تمثّل كل دائرة مساهمة ما commit.



إذا كان في المستودع أربع مساهمات كما يوضّح الشكل أعلاه و أحببنا العودة إلى المساهمة الثانية بشكل خشن hard-- فإنّنا فعليّاً نتخلّى عن المساهمتين التاليتين و يصبح شكل المستودع حالياً كما في الشكل أدناه.



ملحوظة: أنصح بشدّة لمن يرغب بفهم git في العمق أن يمضي بعض الوقت في موقع git visualization حيث يعطيك تصوّراً ممتازاً عمّا يقوم به كل أمر بشكل مرئي، و بالمناسبة، هذا الموقع هو مصدر الشكلين أعلاه.

## استخدام commit --amend

أحياناً و لسبب ما قد نرغب بتغيير ما قمنا بالمساهمة به مؤخّراً، و هو ما يتيحه الأمر git commit --amend، مع التنبيه أنّ هذا الأمّر لا يغيّر المساهمة الأصليّة و إنّما يجمع أيّة موجودة حالياً في بيئة الإدراج staging environment مع آخر مساهمة و ينشئ من كليهما مساهمة جديدة كليّاً، نذكر هذا لأنّ استخدامه يعني تغيير معرّف آخر مساهمة commit ID و لذلك وجب التنبيه.

## git blame حان وقت لوم أحد ما!

أحياناً نرغب و لسبب ما بمعرفة من هو المساهم الذي تسبب بتغيير سطر ما من ملف ما أو مجموعة أسطر، يتيح لنا git ذلك من خلال الأمر git blame و أرى تسميته طريقة لأنّ blame باللغة الإنجليزية تستعمل في الحالات السلبيّة لإلقاء اللوم على أحد ما. على أيّة حال، صيغة الأمر هي:

```
git blame -L <start>,+<end> -- <file-path>
```

حيث أنّ start هو رقم السطر، و end هو عدد الأسطر التالية في حال أنّنا نرغب بالاستعلام عن كتلة أسطر و ليس سطراً واحداً، و file-path هو مسار الملف الذي نستعلم عنه، تظهر الصورة التالية الملامة الواقعة عليّ في تعديل السطر رقم ٣ من الملف index.html.

```
git-book — -zsh — 101×5
Desktop/git-book - (master) > git blame -L 3,+1 -- index.html
^f1d5d3e (Muktar SayedSaleh 2022-06-09 11:15:51 +0300 3)        <title>مرحبا بك في git!</title>
Desktop/git-book - (master) >
```

## git bisect متى حدث هذا؟

أحياناً نرغب و لسبب ما بمعرفة ما هي المساهمة التي تسببت في حدوث مشكلة ما أو تغيير ما في المشروع الذي نعمل عليه، و في حالة المشاريع الكبيرة ذات التاريخ الطويل قد يشكّل هذا تحدّياً إذا كان يعني مراجعة المساهمات واحدة واحدة.

لحسن الحظّ توفّر git آليّة لتوظيف خوارزميّة البحث الثنائي Binary Search في المساهمات للوصول بشكل أسرع بكثير إلى المساهمة المطلوبة و تحديد آلية الاستجابة لها.

و بما أنّني أكتب هذا الكتيّب على نيّة الصدقة و بهدف أن يوزّع إلكترونيّاً و بشكل مجّاني في نهاية المطاف، فسأخالف المعهود قليلاً في مثل هذا السياق، و أحيلك إلى الفيديو التالي الذي يشرح الأمر بمنتهى الروعة و ذلك لأنّ المتعلّم لهذا الأمر يحتاج أن يراه بشكل تفاعلي برأيي.

## و الآن، ماذا بعد؟

لا أزعم أنّني قمت بتغطية كافة الأوامر الموجودة في git، لكنّني أرى أنّني غطّيت القدر الكافي الذي يسمح للمتعلّم الجديد الاستفادة من هذه الأداة في عمله اليومي و أمّا بقية أوامر git فيمكن الرجوع إلى الموقع الرسمي للأداة للحصول على شرح مبسّط يشرح كل أمر من خلال الرابط التالي في حال الحاجة لذلك.

شكراً لكم و الحمد لله الذي منّ علينا بإتمام هذا الكتيّب، أرجو ألا تتردّدوا في التواصل معي في حال الرّغبة، و إلى اللقاء في كتيّبات أخرى قادمة بإذن الله.

تم بحمد الله

إسطنبول ١٠ يونيو ٢٠٢٢

# الفهرس